(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 314 292 B1

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
17.04.1996 Bulletin 1996/16

(51) Int Cl.[6]: **G06F 17/30**

(21) Application number: **88308752.0**

(22) Date of filing: **21.09.1988**

(54) Concurrent record access database system using an index tree structure and method therefor

Datenbanksystem und Verfahren für den gleichzeitigen Satzzugriff mit Hilfe eines Baumstrukturindexes

Système de base de données et méthode pour l'accès concurrent d'enregistrements en utilisant un index à structure d'arbre

(84) Designated Contracting States:
DE FR GB IT

(30) Priority: **30.10.1987 US 115146**

(43) Date of publication of application:
03.05.1989 Bulletin 1989/18

(73) Proprietor: **International Business Machines Corporation**
Armonk, N.Y. 10504 (US)

(72) Inventors:
- **Levine, Frank Eliot**
  Austin Texas 78729 (US)
- **Mohan, Chandrasekaran**
  San Jose California 95121 (US)

(74) Representative: **Bailey, Geoffrey Alan**
IBM United Kingdom Limited
Intellectual Property Department
Hursley Park
Winchester Hampshire SO21 2JN (GB)

(56) References cited:
- **IEEE TRANSACTIONS ON SOFTWARE ENGINEERING. vol. SE-10, no. 6, November 1984, NEW YORK US pages 777 - 781; U. MANBER: 'Concurrent maintenance of binary search trees'**
- **COMPUTING SURVEYS vol. 11, no. 2, June 1979, pages 121 - 137; D. COMER: 'The ubiquitous B-tree'**
- **IBM TECHNICAL DISCLOSURE BULLETIN. vol. 19, no. 10, March 1977, NEW YORK US pages 3887 - 3889; R. BAYER ET AL.: 'Locking protocols for concurrent operations on B-trees'**

EP 0 314 292 B1

Printed by Jouve, 75001 PARIS (FR)

EL302703835US

Description

This invention relates to data processing and more specifically to database management of records.

Current data processing systems include database management programs. These programs provide easy access to database tables that each consist of a multiple of records. A relational database program provides access to several different database tables where elements of one database table are contained in another database table. The relational database program allows the user to search, access, and alter data contained in several different database tables using a specific element or field that is common to these database tables. For example, one database table may contain employees' names and employees' serial numbers. Another database table may contain employees' names and employees' salaries. A third database table may contain employees' names and employees' locations. A relational database program may allow the user to access employees' names, employees' serial numbers, employees' salaries, and employees' locations and make changes to these tables without having to individually access each database table separately.

An important aspect of the database programs is the capability to provide fast and efficient access to records in the individual database.

More recent data processing systems provide support to a multiple of users simultaneously enabling each user to access data concurrently.

An index file is commonly used by database management programs to provide quick and efficient access to records in tables. These index files are commonly configured in a B-Tree structure. A reference that discusses the B-Tree is "Efficient Locking For Concurrent Operation On B-Tree" by Lehman and Yao, ACM Transactions on Database Systems, volume 6, number 4, December, 1981 pages 650-670. Other references addressing B-Tree structures include "The Ubiquitous B-Tree" by Comer, Computing Surveys, volume 11, number 2, June, 1979, pages 121-137; and "Concurrent Operation on B-Trees with Over Taking" by Sagiv, Proceedings ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March, 1985, pages 28-37.

The index file configured as a B-Tree structure consists of a root node with many levels of nodes branching from the root node. The information contained in these nodes include pointers to the nodes at the next level or pointers to records in the database. These pointers include further information termed key record information which may reference the records in the database. The record keys are in an ordered form throughout the nodes. For example, an index tree may exist for an alphabetic listing of employee names. The root node would include reference keyed data that relates to records indirectly or directly referenced by the next level of nodes. The reference keys contain information about the index field, i.e. the alphabetic spelling of the employees name. Therefore, the ordered keys in the root node would point to the next successive level of nodes. In other words, the next successive node may indirectly or directly reference all employees names beginning with A, B, and C.

A next successive node, parallel with the first successive node, may contain employee records whose last name begin with the letters D-M. The last successive node on this level would reference records of employees with last names starting with N-Z. As one searches through the index file tree, a bottom, or leaf, node is eventually reached. The contents of the bottom node include record keys that point to the individual records in storage.

One problem in providing concurrent accesses to database tables occurs when multiple transactions are trying to access a record at the same time. Specifically, when one user wishes to change a record and another user is attempting to access this record, a contention situation occurs. One solution to the contention problem is to provide exclusive access (or locking) to the records or to the portions of the B-Tree indexes to insure that the index node, or record is not changed while the user is attempting to access it. Locking it addressed in "Index Locking and Splitting" IBM Technical Disclosure Bulletin, volume 25, number 7B, December 1982, pages 3725-3729; "Locking Protocols for Concurrent Operation on B-Trees", IBM Technical Disclosure Bulletin, volume 19, number 10, March 1977, pages 3887-3889; and "Concurrent Maintenance of Binary Search Trees", by Manber, IEEE Transactions on Software Engineering, volume SE-10, number 6, November 1984, pages 777-784.

The disadvantage to a locking solution is that a lock, while providing access to cne user, prevents access by any other user.

Information in index nodes may be altered during access by several users. These nodes contain the key record information that may be deleted or inserted into or deleted from the database table. Information from one node may be moved to another node during a record insert operation because of limited storage space for a node. The inserting transaction may attempt to move key record information from one node to another in order to accomplish the insertion task. This movement of information during concurrent transaction access may generate error conditions within the concurrent transaction accesses because the index information is being changed.

It is an object of the present invention to provide concurrent access to the index by several transactions while the index information is being changed without introducing error conditions.

According to the invention, there is provided a method of operating a computer system for effecting simultaneous transactions on a database by multiple users using an index configured as a tree structure having a root node and lower level nodes, each node, except those in lowest level leaf nodes, referencing one or more nodes in the next lower level, with leaf nodes referencing cor-

responding records in the database, and each node, except leaf nodes, including key information to identify the next level node to be accessed in searching for a particular record, and each leaf node including keys relating to records referenced by that leaf node, the keys being ordered throughout the leaf nodes, said method comprising, for a transaction involving insertion or deletion of a current record in the database and its corresponding key in a leaf node in the index, the steps of:

traversing through the nodes of the tree from said root node to the leaf node corresponding to said current record in response to an input request to access said current record, using the key information in said nodes;
limiting access to a node being traversed and an immediately previously traversed node such that other users are only allowed read access to said node being traversed and said immediately previously traversed node;
inserting or deleting said current record and its corresponding record key; and removing, after said step of inserting or deleting, all of the access limitations to the traversed nodes;
and characterised by the steps of:
prior to said step of inserting or deleting, limiting access to said leaf node such that no other user may access said leaf node, identifying a next successively located key in terms of the ordering of keys, relative to the key of the current record to be deleted or inserted, and limiting access to the record corresponding to said next successively located key such that no other user may access said record; and
removing, after said step of inserting or deleting, the access limitation to the record corresponding to the next successively located key.

There is further provided a digital data processing system for effecting simultaneous transactions on a database by multiple users, including an index configured as a tree structure having a root node and lower level nodes, in which each node, except those in lowest level leaf nodes, references one or more nodes in the next lower level, with leaf nodes referencing corresponding records in the database, and in which each node, except leaf nodes, includes key information to identify the next level node to be accessed in searching for a particular record, with each leaf node including keys relating to records referenced by that leaf node, the keys being ordered throughout the leaf nodes, said system further including means for processing a transaction involving insertion or deletion of a current record in the database and its corresponding key in a leaf node in the index, said processing means comprising:

means for traversing through the nodes of the tree from said root node to the leaf node corresponding to said current record in response to an input request to access said current record, using the key information in said nodes;
means for limiting access to a node being traversed and an immediately previously traversed node such that other users are only allowed read access to said node being traversed and said immediately previously traversed node;
means for inserting or deleting said current record and its corresponding record key; and means for removing, after said inserting or deleting, all of the access limitations to the traversed nodes;
and characterised by:
means for limiting access to said leaf node such that no other user may access said leaf node, for identifying a next successively located key in terms of the ordering of keys, relative to the key of the current record to be deleted or inserted, and for limiting access to the record corresponding to said next successively located key such that no other user may access said record, prior to said inserting or deleting; and
means for removing, after said inserting or deleting, the access limitation to the record corresponding to the next successively located key.

An embodiment of the invention will now be described, by way of example, with reference to the accompanying drawings, in which:

Figure 1 is a illustration of a database table;
Figure 2 is a illustration of the database table stored in a computer memory;
Figure 3 is a block diagram illustrating a simple B-Tree index;
Figure 4 is an illustration of the storage of index B-Tree node data in a computer memory;
Figure 5 is a flow chart illustrating an initial search through an index B-Tree;
Figure 6 is a flow chart illustrating a fetch through the index B-Tree;
Figure 7 is a flow chart illustrating an insert through the index B-Tree;
Figure 8 is a flow chart illustrating a delete through the index B-Tree;
Figure 9 is a flow chart illustrating a node splitting algorithm; and
Figure 10 is a flow chart illustrating a node collapsing algorithm.

This invention relates to the accessing of records in a database system. The following example is an overly simplified database access problem provided to enable the reader to more easily understand the invention contained herein. The reader should understand, that, in reality, the number of entries and amount of information that is contained in database tables are much greater than illustrated.

Figure 1 illustrates table 10 containing employee

names and numbers. Each employee record includes a single employee number. The names in table 10 are listed in alphabetical order. However, it should be apparent to those skilled in the art that this is not how the records are normally stored in the computer memory.

Figure 2 is an illustration of the storage of the employee name/number key records in a computer memory 12. Each record 14, 16, 18, 20, 24, and 26 consists of several portions. In record 14, the record first consist of an address portion 28, the employee name 30, and the employee number 32. The address 28 is the actual location of the employee name 30 and the employee number 32 in the computer memory. Table 12 of Figure 2 illustrates that the storage of the names is not in alphabetical order and is not even contiguous.

The database system normally provides an index tree that enables a user to quickly access a specific record in storage. An example of such an index is provided in Figure 3. For simplicity of presentation, the index tree is presented in two levels. Index 40 consist of a root node connected to three nodes 50, 52, and 54. The root node 42 is termed "index identifier" and details the type of information in the index. In this example, the index is an alphabetically ordered (ascending) index of the employees' names. Each employee's name includes a key. In this example, the key is the first letter of the employee's name. The keys are used to identify a next level node to be accessed in searching for a specific employee name record. In the root node 42, the keys consist of letters "D" 44,"F" 46, "NULL" 48. The D key 44 directs the user from the root node 42 (the parent node) to node 50 (a child node). Node 50 contains key record information for direct access of the employees Andrews, Baker, and Chester. However, only a single key is used in the root node. This key relates to the highest level of information (or the highest key record) that might now exist or have existed in the node 50. Likewise the "F" 46 directs a user to node 52 that contains the Edgar and Edwards employee names. Lastly, "NULL" 48 directs the user to node 54. The "NULL" key indicates that node 54 is the last successive leaf node. It should be apparent to those skilled in the art that the index 40 may also contains multiple levels of nodes. The two level index 40 is provided for the sake of simplicity.

The contents of node 50, 52 and 54 as it would appear in a computer memory is illustrated in Figure 4 as columns 56, 58 and 60 respectively. Column 56 represents how the information contained in node 50 would be stored in a computer memory. Column 56 consist of a set of pointers 62 and data storage area 64. Pointer 66 points to the information containing the employee name Andrew and employee number. Likewise, pointer 68 points to the employee name Baker and related information. Thus, when a transaction accesses a node, the transaction first examines the pointer portion of the node information to determine where the key record information is stored. This pointer information provides direct access to the storage containing the desired information. In multilevel indexing structure including intermediate levels, the node storage would contain pointers that point to addresses for the next successively located nodes according the key record information. The pointers would be ordered according to their respective key information. However, a successive node, such as 50 that contains several employee names is only referenced by a single key from a parent node. The reference key is the highest key record information allowable in the node. In index trees including intermediate nodes, the key in a preceding (or parent) node is a key for the highest record that is referenced by any of the keys.

Figure 5 illustrates a flow chart for accessing record information through the index. It should be understood that in a database management system, the operations illustrated by Figures 5-8 is called during an access transaction. In accordance with the illustrated example, the transaction would be attempting to fetch a record to determine the employee number or to perform some other operation such as record insert or record delete operation.

Figures 3 and 4 also illustrate pointers 51 and 53 that point from one leaf node to a next successively located leaf node. These pointers 72 and 74 direct an accessing transaction to the next successively located leaf node. In this example, the NULL entry 76 in node 60 indicates that node 60 is the last node.

In Figure 5, the access logic starts at step 100 and proceeds to step 102 where the index root node is S-Latched and accessed. An S-Latch provides limited access to other concurrent users. This limited access provides the other users with the capability to only read the node. No other access, such as the capability to delete or change, is provided. The index identifier node identifies the type of index and provides the initial direction for accessing a record in this example. The index identifier node would identify the index as an alphabetic index in ascending order for the employee names. In step 103, the child node to be accessed would be identified according to the information in the parent node. In step 104, it is determined whether not the operation to be performed is a fetch operation. If the operation is not a fetch operation, in other words the operation is a record insert (or key record insert) or record delete (or key record delete) operation, the program proceeds to step 106 to determine if the next node beneath the parent node is a leaf node or bottom node. If the next node is a leaf node, the program proceeds to the step 110 and acquires an X-Latch on the child node. The X-Latch is an exclusive latch that excludes all other accesses to this node. In other words, the transaction that applies the X-Latch excludes all other transactions from accessing this node.

Returning to step 104, if the operation is a fetch operation or, returning to step 106, if the child is not a leaf node, the logic proceeds to step 108 to acquires an S-Latch on the child node. The logic proceeds to step 112, where it accesses the child node and determines if

the key of the record being searched is greater than the key in the child node. If this key is greater, the logic proceeds to step 114 to determine if the tree index structure is latched. If the tree is not latched, the logic proceeds to step 118 wherein the parent and child nodes are unlatched and the tree is latched. The logic then proceeds from step 118 back to step 102 to reinitiate the operation upon the granting of the tree latch. Optimizations are possible to reduce the number of nodes to be accessed when the operation is re-attempted.

In this example, an X-Latch on a tree is provided to indicate to all other accesses that a change in tree structure is being made. If a tree X-Latch access is in progress when a latch is attempted on the tree, the attempting access must wait till the earlier access is complete. An S-Latch on a tree is provided to all other accesses to indicate that no structure changes are being made but other accesses may concurrently access the index tree. No other changes can occur until the S-Latch is released. Tree traversal may occur regardless of the existence of S-Latches or X-Latches. These tree traversals may include key record deletions or insertions.

In step 114 if the tree is latched, or in step 112 if the key is not greater than the highest key in the child, the logic proceeds to step 116 to determine if the child is a leaf. If the child is not, the logic proceeds to step 115 to unlatch the parent node and then returns to step 103. However, if the child is a leaf, the logic proceeds to step 120 to unlatch the parent and then to step 122, 124 and 126 to determine if the operation is a fetch, insert or delete operation. In this example provided, if neither of these three operations are attempted, the logic will return to the user in step 130. In practice this return would include an error indicator signifying that the operation to be performed is not identifiable by this accessing attempts

If the operation is a fetch operation, the logic proceeds to step 200 illustrated on Figure 6. In step 200, the logic finds the requested key being searched or at least the next highest key. In step 202, the logic then requests a conditional lock on the key record. In this example a conditional lock is requested from managing logic that manages the locks on the record keys. The term "conditional" means that if the lock is not immediately granted, a response will be provided to the requesting accessor indicating that such a lock is not being granted. This response is used in Step 204. If the lock has not been granted, the logic proceeds to step 206 to unlatch the child node and then to step 208 to request an unconditional lock on the key record. In step 208, the accessor then waits until such lock is granted before it proceeds. Once the lock is granted, it proceeds though connector 213 to step 102 in Figure 5 to restart the search. Returning to step 204, if the lock has been granted, the logic proceeds to step 209 to return the key record data. In step 210, the child node is unlatched. In step 211, the key record is unlocked. It should be apparent to those skilled in the art that step 211 will occur when the transaction is complete or at an earlier time.

In Figure 7, the insert operation is illustrated. It should be apparent to those skilled in the art that the key record to be inserted will be X-Locked, if necessary, before the insert operation begins. This operation would insert a record in the computer memory and provide an inserted key into the index updating the respective node (s) of the index enabling other transactions to access the newly inserted record. In step 300, it is first determined if the key can fit into the bottom or leaf node. If so, the logic proceeds to step 304 to find the next key that is greater than the key to be inserted. If the next key is not in the node, the access is pointed to the next successively located leaf node. If there is no next successively located leaf node, the access finds the null indicator. Then, in step 308, the logic request a conditional X-Lock on the next key record. In step 312, it is then determined whether or not this lock has been granted. If so, in step 318 the new key is inserted on the first bottom node that was reached in step 300. In step 324, the next record and key record are unlocked. Also, any held latches are released in step 329. Then, in step 325, the logic returns. It should be apparent to those skilled in the art that the lock on the inserted key record will be released when the transaction is complete. Returning to step 312, if the lock is not granted, the logic proceeds to step 316 to unlatch the child and then to step 322 to request an unconditional X-Lock on the next key record. The logic then returns through connector 328 to step 102 (Figure 5).

Returning to step 300, if the key will not fit into the leaf node, the logic proceeds to step 302 to request a conditional tree X-Latch. In step 306, it is determined whether or not the conditional latch is granted. If not the logic proceeds to step 310 to unlatch the child node and then to step 314 to request an unconditional tree X-Latch. Upon being granted the unconditional tree X-Latch or, in step 306 if the latch is granted, the logic proceeds to step 320 to perform a node splitting algorithm. This node splitting algorithm provides a new node containing a portion of the information from the previous node. The node splitting algorithm may also provide additional intermediate nodes wherein the additional nodes would be updated. The parent node is modified to contain the highest key record and a pointer to the new node. A node splitting algorithm is illustrated in Figure 9. Furthermore, node splitting is addressed in Data Structures and Algorithms, by Aho, Hopcroft and Ullman, Addison-Wesley Publishing Company, 1983, pages 170-179. Upon completion of the node splitting algorithm, the logic proceeds to step 326 to unlatch the tree. The logic would then return to step 102 (Figure 5) through connector 327.

Figure 8 illustrates the delete operation. It should be apparent to those skilled in the art that the key record to be deleted will be X-Locked, if necessary, before the delete operation begins. In step 400, the logic finds the next key greater than the key to be deleted. In step 402, the logic requests a conditional X-Lock on this next key record. The logic proceeds to step 404 to determine if

the lock is granted. If the lock is not granted, the logic proceeds to unlatch the child node in step 406 and then to request an unconditional lock on the next key record in step 410. The logic then proceeds through connector 431 to step 102 (Fig. 5).

Returning to step 404, if the lock is granted, then the key is deleted in step 408. The logic then proceeds to step 412 to determine if the next node is empty. If not, the logic unlatches the node in step 421, unlocks the next key record in step 413 and returns in step 414. However, if the node is then empty, the logic proceeds to step 416 to request a conditional tree X latch. Then in step 418 the program determines if the latch has been granted. If not, in step 420, the logic unlatches the child node and then in step 422, proceeds to request an unconditional tree X latch. Once the latch is granted, either from step 422 or from step 418, the logic proceeds to step 424 to perform the node collapsing algorithm which removes the empty node and the references to the empty node from the preceding nodes in the index tree. This algorithm is illustrated in Figure 10. In step 426 the tree is unlatched. In step 429, the deleted key record and the next key record is unlocked. It should be apparent to those skilled in the art that step 429 will occur when the transaction is complete.

Figure 9 illustrates a node splitting algorithm. In this algorithm a new node is added to the index tree to store the newly inserted key record. Additionally all the preceding, or parent, nodes in the index tree may be updated to reflect the addition of this new node and the information contained therein. In step 500, a new node is obtained. In step 502, the new node is X-Latched. In step 504 a portion of the keys from the node that is being split is moved to the new node. It should be apparent that a pointer is also included in the node being split to reference the new node. Additionally, if required, a pointer in the new node is provided to indicate the next successively located node to the new node. In step 506, the node being split and the new node are both unlatched. In step 507, the parent node to the node being split is then X-Latched. In step 508, the parent node is updated. In step 510 the parent node is unlatched. It should be apparent to those skilled in the art, that the updating of the parent node may actually include the splitting of the parent node and the updating of several successively located parent nodes up the index tree.

Figure 10 illustrates a node collapsing algorithm. The purpose of the node collapsing algorithm is to remove an empty node from the index tree to conserve memory and to provide efficient access to keys. In step 600, a node adjacent to the empty node is X-Latched. In step 602, the keys of this adjacent node are then moved to the empty node. In step 604, the pointer in the receiving node the keys is then updated to locate the node with newly loaded keys. In step 606, the next node, now empty is removed. In step 608, the latches for the child node, the node with the newly removed keys, and the adjacent node, that has been removed, are unlatched. The parent node or nodes are then X-Latched in step 610. In step 612, the parent node or nodes are updated to remove the pointer to the removed node and to adjust the key associated with the node that receives the keys from the removed node. In step 614, the parent nodes are then unlatched. As discussed in Figure 9, the updating of the parent node or nodes maybe a recursive process as would be apparent to those skilled in the art.

The present system provides for a number of database transactions to access an index tree concurrently. This concurrent access is provided in a manner that guarantees that repeated accesses will provide consistent results within a single transaction if deemed necessary. In other words, the procedure previously illustrated provides that a transaction that is accessing an index tree during the change of index tree by another transaction will not return erroneous or invalid information that may result from accessing a portion of the index tree during this change. This is possible because of the tree latch. The tree latch is used in a way to allow concurrent accesses to the tree during the index tree change. However, the transaction protocol requires that when a condition exists wherein an error is possible, the transaction encountering this possible error condition requests a latch on the tree as opposed to returning an error condition or erroneous data. By requesting a latch on the tree, the requesting transaction must wait if another transaction that, is in fact, changing the index tree structure because this earlier transaction would have also requested a tree latch. When the tree latch is finally granted, the requesting transaction will then be assured that no changes are being made to the index tree structure during its access. This procedure is advantageous over excluding access to the index tree in that it allows concurrent accesses and provides a mechanism to insure that transaction will not retrieve information that is being changed during an index tree structure change.

Claims

1. A method of operating a computer system for effecting simultaneous transactions on a database by multiple users using an index (40) configured as a tree structure having a root node (42) and lower level nodes (50, 52, 54), each node, except those in lowest level leaf nodes, referencing one or more nodes in the next lower level, with leaf nodes referencing corresponding records in the database, and each node, except leaf nodes, including key information (44, 46, 48) to identify the next level node to be accessed in searching for a particular record, and each leaf node including keys relating to records referenced by that leaf node, the keys being ordered throughout the leaf nodes, said method comprising, for a transaction involving insertion or deletion of a current record in the database and its corresponding key in a leaf node in the index, the steps of:

traversing (103, 104) through the nodes of the tree from said root node to the leaf node corresponding to said current record in response to an input request to access said current record, using the key information in said nodes;
limiting access (108, 115) to a node being traversed and an immediately previously traversed node such that other users are only allowed read access to said node being traversed and said immediately previously traversed node;
inserting (318) or deleting (408) said current record and its corresponding record key; and
removing (329, 421), after said step of inserting or deleting, all the access limitations to the traversed nodes;
and characterised by the steps of:
prior to said step of inserting or deleting, limiting access (110) to said leaf node such that no other user may access said leaf node, identifying (304; 400) a next successively located key in terms of the ordering of keys, relative to the key of the current record to be deleted or inserted, and limiting access (308, 402) to the record corresponding to said next successively located key such that no other user may access said record; and
removing (324, 413), after said step of inserting or deleting, the access limitation to the record corresponding to the next successively located key.

2. A method according to claim 1, in which, for inserting said current record, the removal of the access limitation to the record associated with the next successively located key precedes the removal of all access limitations to the traversed nodes.

3. A method according to claim 1, in which, for deleting said current record, the removal of all access limitations to the traversed nodes precedes the removal of the access limitation to the record associated with the next successively located key.

4. A method according to claim 1 or 2 including, for inserting said current record, the steps of:

determing if the key associated with the current record can be inserted in said leaf node; and

if the key can be so inserted, inserting the key and the current record; or

if the key cannot be so inserted:

providing an indication that the index tree structure is to be changed;

altering the index tree to add a node to receive said current record;

removing the indication of the tree structure change; and

inserting the key and the current record.

5. A method according to claim 1 or 3 including, for deleting said current record, the steps of:

determining if the key associated with the current record is the only record in said leaf node; and

if it is not, deleting the key and the current record; or, if it is: providing an indication that the index tree structure is to be changed;

deleting the key and the current record;

altering the index tree structure to remove the node; and

removing the indication of the index tree structure change.

6. A digital data processing system for effecting simultaneous transactions on a database by multiple users, including an index (40) configured as a tree structure having a root node (42) and lower level nodes (50, 52, 54), in which each node, except those in lowest level leaf nodes, references one or more nodes in the next lower level, with leaf nodes referencing corresponding records in the database, and in which each node, except leaf nodes, includes key information (44, 46, 48) to identify the next level node to be accessed in searching for a particular record, with each leaf node including keys relating to records referenced by that leaf node, the keys being ordered throughout the leaf nodes, said system further including means for processing a transaction involving insertion or deletion of a current record in the database and its corresponding key in a leaf node in the index, said processing means comprising:

means for traversing through the nodes of the tree from said root node to the leaf node corresponding to said current record in response to an input request to access said current record, using the key information in said nodes;
means for limiting access to a node being traversed and an immediately previously traversed node such that other users are only allowed read access to said node being traversed and said immediately previously traversed node;
means for inserting or deleting said current record and its corresponding record key; and

means for removing, after said inserting or deleting, all of the access limitations to the traversed nodes;
and characterised by:
means for limiting access to said leaf node such that no other user may access said leaf node, for identifying a next successively located key in terms of the ordering of keys, relative to the key of the current record to be deleted or inserted, and for limiting access to the record corresponding to said next successively located key such that no other user may access said record, prior to said inserting or deleting; and
means for removing, after said inserting or deleting, the access limitation to the record corresponding to the next successively located key.

7. A system according to claim 6 including, for inserting said current record, means for determining if the key associated with the current record can be inserted in said leaf node, and, for when the key can not be so inserted, means for providing an indication that the index tree structure is to be changed, means for altering the index tree to add a node to receive said current record, and for removing the indication of the tree structure change prior to insertion of the key and the current record.

8. A system according to claim 6 including, for deleting said current record, means for determining if the key associated with the current record is the only record in said leaf node and, for when it is, means for providing an indication that the index tree structure is to be changed means for deleting the key and the current record, means for altering the index tree structure to remove the node, and for removing the indication of the index tree structure change.

Patentansprüche

1. Ein Verfahren zum Betreiben eines Rechnersystems zur gleichzeitigen Durchführung von Transaktionen in einer Datenbank durch Mehrfach-Anwender unter Verwendung eines Index (40), der als Baumstruktur konfiguriert ist, die einen Wurzelknoten (42) und Knoten (50, 52, 54) der unteren Ebene aufweist, wobei jeder Knoten, abgesehen von denen in der untersten Ebene, Blattknoten sind, die einen oder mehrere Knoten in der nächsttieferen Ebene ansprechen und die Blattknoten entsprechende Datensätze in der Datenbank ansprechen, und jeder Knoten, abgesehen von den Blattknoten, Schlüsselinformationen (44, 46, 48) beinhaltet zum identifizieren des Knotens der jeweils nächsten Ebene zum Zugriff darauf bei der Suche nach einem bestimmten Datensatz, und jeder Blattknoten Schlüssel enthält, die sich auf die von diesem Blattknoten angesprochenen Knoten beziehen, wobei die Schlüssel durch die Blattknoten geordnet sind, wobei dieses Verfahren im Hinblick auf eine Transaktion, die das Einschieben oder das Löschen eines augenblicklichen Datensatzes in der Datenbank und deren entsprechenden Schlüssel in einem Blattknoten im Index, die folgenden Schritte beinhaltet:

Durchlaufen (103, 104) der Knoten des Baums vom Wurzelknoten zum Blattknoten, der dem laufenden Datensatz entspricht als Reaktion auf eine Eingabeanforderung, auf den augenblicklichen Datensatz Zugriff zu nehmen unter Verwendung der Schlüsselinformationen in diesen Knoten;

Einschränken des Zugriffs (108, 115) auf einen Knoten, der durchlaufen wird, sowie auf den unmittelbar davor durchlaufenen Knoten, so daß andere Anwender nur zum Lesezugriff auf den eben durchlaufenen Knoten und auf den unmittelbar vorher durchlaufenen Knoten zugelassen sind;

Einschieben (318) bzw. Löschen (408) des aktuellen Datensatzes und seiner entsprechenden Datensatzschlüssel;

und nach dem Schritt des Einschiebens bzw. Löschens Entfernen (329, 421) aller Zugriffsbeschränkungen auf die durchlaufenen Knoten;

gekennzeichnet durch die folgenden Schritte:

vor dem Einfüge- bzw. Löschschritt Einschränken des Zugriffs (110) auf den Blattknoten, so daß kein anderer Anwender auf diesen Blattknoten Zugriff haben kann, Identifizieren (304, 400) eines nächsten nachfolgend angeordneten Schlüssels in der Form der Schlüsselreihenfolge bezüglich des Schlüssels des laufenden Datensatzes, der gelöscht oder eingefügt werden soll, und Einschränken des Zugriffs (308, 402) auf den Datensatz entsprechend diesem nächsten nachfolgend angeordneten Schlüssels, so daß kein anderer Anwender auf diesen Datensatz Zugriff nehmen kann; und

nach diesem Schritt des Einschiebens oder Löschens Entfernen (324, 413) der Zugriffseinschränkung auf den Datensatz entsprechend dem nächsten nachfolgend angeordneten Schlüssel.

2. Ein Verfahren gemäß Anspruch 1, in dem, zwecks Einfügen des laufenden Datensatzes, das Entfernen der Zugriffseinschränkung für den Datensatz,

der dem nächstfolgenden Schlüssel zugeordnet ist, dem Entfernen aller Zugriffseinschränkungen auf die durchlaufenen Knoten vorhergeht.

3. Ein Verfahren gemäß Anspruch 1, in dem, zwecks Löschens des laufenden Datensatzes, das Entfernen aller Zugriffseinschränkungen auf die durchlaufenen Knoten dem Entfernen der Zugriffseinschränkungen auf den Datensatz, der dem nächstfolgenden Schlüssel zugeordnet ist, vorhergeht.

4. Ein Verfahren gemäß Anspruch 1 oder 2, einschließlich, zwecks Einfügens des augenblicklichen Datensatzes, der folgenden Schritte:

Feststellen, ob der dem augenblicklichen Datensatz zugeordnete Schlüssel in den Blattknoten eingefügt werden kann; und

wenn der Schlüssel eingefügt werden kann, Einfügen des Schlüssels und des augenblicklichen Datensatzes; oder

wenn der Schlüssel nicht eingefügt werden kann:

Vorsehen einer Anzeige, daß die Indexbaumstruktur geändert werden muß;

Verändern des Indexbaums, um einen Knoten zur Aufnahme des augenblicklichen Datensatzes hinzuzufügen;

Entfernen der Anzeige über die Baumstrukturveränderung; und

Einfügen des Schlüssels und des augenblicklichen Datensatzes.

5. Ein Verfahren gemäß Anspruch 1 oder 3, einschließlich, zwecks Löschens des augenblicklichen Datensatzes, der folgenden Schritte:

Feststellen, ob der dem augenblicklichen Datensatz zugeordnete Schlüssel der einzige Datensatz im Blattknoten ist; und

wenn nicht, Löschen des Schlüssels und des augenblicklichen Datensatzes; oder wenn er es ist:

Vorsehen einer Anzeige, daß die Indexbaumstruktur geändert werden muß;
Löschen des Schlüssels und des augenblicklichen Datensatzes;
Verändern der Indexbaumstruktur, um den Knoten zu entfernen; und
Entfernen der Anzeige der Indexbaumstrukturänderung.

6. Ein digitales Datenverarbeitungssystem zum gleichzeitigen Durchführen von Transaktionen in einer Datenbank durch Mehrfach-Anwender einschließlich eines Index (40), der als Baumstruktur konfiguriert ist, die einen Wurzelknoten (42) und Knoten (50, 52, 54) der unteren Ebene aufweist, wobei jeder Knoten, abgesehen von den Blattknoten der untersten Ebene einen oder mehrere Knoten in der nächsttieferen Ebene ansprechen und die Blattknoten entsprechende Datensätze in der Datenbank ansprechen, und in denen jeder Knoten, abgesehen von den Blattknoten, Schlüsselinformationen (44, 46, 48) beinhaltet zum Identifizieren des Knotens der jeweils nächsten Ebene zum Zugriff darauf bei der Suche nach einem bestimmten Datensatz, und jeder Blattknoten Schlüssel enthält, die sich auf die von diesem Blattknoten angesprochenen Knoten beziehen, wobei die Schlüssel durch die Blattknoten geordnet sind, wobei dieses System ferner Mittel zum Durchführen einer Transaktion zum Einschieben bzw. Löschen eines augenblicklichen Datensatzes in der Datenbank und dessen entsprechender Schlüssel in einem Blattknoten im Index beinhaltet, wobei dieses Verarbeitungsmittel umfaßt:

Mittel zum Durchlaufen der Knoten des Baums vom Wurzelknoten zum Blattknoten, der dem laufenden Datensatz entspricht, als Reaktion auf eine Eingabeanforderung, auf den laufenden Datensatz Zugriff zu nehmen unter Verwendung der Schlüsselinformationen in diesen Knoten;

Mittel zum Einschränken des Zugriffs auf einen Knoten, der durchlaufen wird, sowie den unmittelbar davor durchlaufenen Knoten, so daß andere Anwender nur zum Lesezugriff auf den eben durchlaufenen Knoten und auf den unmittelbar vorher durchlaufenen Knoten zugelassen sind;

Mittel zum Einschieben bzw. Löschen des aktuellen Datensatzes und seiner entsprechenden Datensatzschlüssel; und

Mittel zum Entfernen aller Zugriffsbeschränkungen auf die durchlaufenen Knoten nach dem Schritt des Einschiebens bzw. Löschens;

und gekennzeichnet durch:

Mittel zum Einschränken des Zugriffs auf den Blattknoten, so daß kein anderer Anwender auf diesen Blattknoten Zugriff haben kann, zum Identifizieren eines nächsten nachfolgend angeordneten Schlüssels in der Form der

Schlüsselreihenfolge bezüglich des Schlüssels des laufenden Datensatzes, der gelöscht bzw. eingefügt werden soll, und zum Einschränken des Zugriffs auf den Datensatz entsprechend diesem nächsten nachfolgend angeordneten Schlüssel, so daß kein anderer Anwender vor dem Einfüge- bzw. Löschschritt auf diesen Datensatz Zugriff nehmen kann; und

Mittel zum Entfernen der Zugriffseinschränkung auf den Datensatz entsprechend dem nächsten nachfolgend angeordneten Schlüssel nach diesem Schritt des Einschiebens oder Löschens.

7. Ein System gemäß Anspruch 6, einschließlich, zwecks Einschiebens des augenblicklichen Datensatzes, Mittel zum Festlegen, ob der dem augenblicklichen Datensatz zugeordnete Schlüssel in den Blattknoten eingefügt werden kann, und, wenn der Schlüssel nicht eingefügt werden kann, Mittel zum Vorsehen einer Anzeige, daß die Indexbaumstruktur geändert werden muß, Mittel zum Ändern des Indexbaums, um einen Knoten einzufügen, um den augenblicklichen Datensatz aufzunehmen, und zum Entfernen der Anzeige der Baumstrukturänderung vor dem Einfügen des Schlüssels und des augenblicklichen Datensatzes.

8. Ein System gemäß Anspruch 6, einschließlich, zwecks Löschens des augenblicklichen Datensatzes, Mittel zum Festlegen, ob der dem augenblicklichen Datensatz zugeordnete Schlüssel der einzige Datensatz in dem Blattknoten ist und, wenn er es ist, Mittel zum Vorsehen einer Anzeige, daß die Indexbaumstruktur geändert werden muß, Mittel zum Löschen des Schlüssels und des augenblicklichen Datensatzes, Mittel zur Ändern der Indexbaumstruktur zwecks Entfernens des Knotens und zwecks Entfernens der Anzeige über die Indexbaumstrukturänderung.

Revendications

1. Méthode d'exploitation d'un système informatique pour effectuer des transactions simultanées sur une base de données par de multiples utilisateurs utilisant un index (40) dont la configuration a la forme d'une structure arborescente ayant un noeud de racine (42) et des noeuds de niveau inférieur (50, 52, 54), chaque noeud, à l'exception de ceux dans des noeuds de feuille de niveau inférieur, indiquant un ou plusieurs noeuds dans le prochain niveau inférieur, des noeuds de feuille indiquant des enregistrements correspondants dans la base de données, et chaque noeud, à l'exception des noeuds de feuille, comprenant des informations de clé (44, 46, 48) pour identifier le prochain noeud de niveau auquel il doit y avoir accès dans la recherche d'un enregistrement particulier, et chaque noeud de feuille comprenant des clés relatives à des enregistrements indiqués par ce noeud de feuille, les clés étant ordonnées dans les noeuds de feuille, ladite méthode comprenant, pour une transaction impliquant l'insertion ou l'effacement d'un enregistrement en cours dans la base de données et de sa clé correspondante dans un noeud de feuille de l'index, les étapes de:

parcourir (103, 104) les noeuds de l'arbre à partir dudit noeud de racine jusqu'au noeud de feuille correspondant audit enregistrement en cours en réponse à une requête d'entrée pour un accès audit enregistrement en cours, en utilisant les informations de clé dans lesdits noeuds;
limiter l'accès (108, 115) à un noeud parcouru et à un noeud parcouru immédiatement avant pour que d'autres utilisateurs ne puissent qu'avoir un accès de lecture audit noeud parcouru et audit noeud parcouru immédiatement avant;
insérer (318) ou effacer (408) ledit enregistrement en cours et sa clé d'enregistrement correspondante; et
supprimer (329, 421) après ladite étape d'insérer ou d'effacer, toutes les limitations d'accès aux noeuds parcourus;
et caractérisée par les étapes de:
avant ladite étape d'insérer ou d'effacer, limiter l'accès (110) audit noeud de feuille pour qu'aucun autre utilisateur ne puisse avoir accès audit noeud de feuille, identifier (304, 400) une prochaine clé suivante en fonction de l'ordonnancement des clés, par rapport à la clé de l'enregistrement en cours à effacer ou insérer, et limiter l'accès (308, 402) à l'enregistrement correspondant à ladite prochaine clé suivante pour qu'aucun autre utilisateur ne puisse avoir accès audit enregistrement; et
supprimer (324, 413), après ladite étape d'insérer ou d'effacer, la limitation d'accès à l'enregistrement correspondant à la prochaine clé suivante.

2. Méthode selon la revendication 1 dans laquelle, pour insérer ledit enregistrement en cours, la suppression de la limitation d'accès à l'enregistrement associé à la prochaine clé suivante, précède la suppression de toutes les limitations d'accès aux noeuds parcourus.

3. Méthode selon la revendication 1 dans laquelle, pour effacer ledit enregistrement en cours, la suppression de toutes les limitations d'accès aux noeuds parcourus, précède la suppression de la

limitation d'accès à l'enregistrement associé à la prochaine clé suivante.

4. Méthode selon les revendications 1 ou 2 comprenant, pour insérer ledit enregistrement en cours, les étapes de:

déterminer si la clé associée à l'enregistrement en cours peut être insérée dans ledit noeud de feuille; et
si la clé peut être insérée ainsi:

insérer la clé et l'enregistrement en cours; ou

si la clé ne peut pas être insérée:

fournir une indication que la structure arborescente de l'index doit être changée;
modifier l'arbre d'index afin d'ajouter un noeud pour recevoir ledit enregistrement en cours;
supprimer l'indication du changement de la structure arborescente; et
insérer la clé et l'enregistrement en cours.

5. Méthode selon les revendications 1 ou 3 comprenant, pour effacer ledit enregistrement en cours, les étapes de:

déterminer si la clé associée à l'enregistrement en cours est le seul enregistrement dans ledit noeud de feuille; et
dans la négative, effacer la clé et l'enregistrement en cours; ou, dans l'affirmative:

fournir une indication que la structure arborescente d'index doit être changée;
effacer la clé et l'enregistrement en cours;
modifier la structure arborescente d'index pour supprimer le noeud; et
supprimer l'indication du changement de la structure arborescente d'index.

6. Système de traitement de données numériques pour effectuer des transactions simultanées sur une base de données par de multiples utilisateurs utilisant un index (40) dont la configuration a la forme d'une structure arborescente ayant un noeud de racine (42) et des noeuds de niveau inférieur (50, 52, 54), dans lequel chaque noeud, à l'exception de ceux dans des noeuds de feuille de niveau inférieur, indique un ou plusieurs noeuds dans le prochain niveau inférieur, des noeuds de feuille indiquant des enregistrements correspondants dans la base de données, et dans lequel chaque noeud, à l'exception des noeuds de feuille, comprend des informations de clé (44, 46, 48) pour identifier le prochain noeud de niveau auquel il doit y avoir accès dans la recherche d'un enregistrement particulier, chaque noeud de feuille comprenant des clés relatives à des enregistrements indiqués par ce noeud de feuille, les clés étant ordonnées dans les noeuds de feuille, ledit système comprenant en outre des moyens de traitement pour traiter une transaction impliquant l'insertion ou l'effacement d'un enregistrement en cours dans la base de données et de sa clé correspondante dans un noeud de feuille de l'index, lesdits moyens de traitement comprenant:

des moyens pour parcourir les noeuds de l'arbre à partir dudit noeud de racine jusqu'au noeud de feuille correspondant audit enregistrement en cours en réponse à une requête d'entrée pour un accès audit enregistrement en cours, en utilisant les informations de clé dans lesdits noeuds;
des moyens pour limiter l'accès à un noeud parcouru et à un noeud parcouru immédiatement avant pour que d'autres utilisateurs ne puissent qu'avoir un accès de lecture audit noeud parcouru et audit noeud parcouru immédiatement avant;
des moyens pour insérer ou effacer ledit enregistrement en cours et sa clé d'enregistrement correspondante; et
des moyens pour supprimer, après ladite étape d'insérer ou d'effacer, toutes les limitations d'accès aux noeuds parcourus;
et caractérisé par:
des moyens pour limiter l'accès audit noeud de feuille pour qu'aucun autre utilisateur ne puisse avoir accès audit noeud de feuille, pour identifier une prochaine clé suivante en fonction de l'ordonnancement des clés, par rapport à la clé de l'enregistrement en cours à effacer ou insérer, et pour limiter l'accès à l'enregistrement correspondant à ladite prochaine clé suivante pour qu'aucun autre utilisateur ne puisse avoir accès audit enregistrement avant ladite étape d'insérer ou d'effacer; et
des moyens pour supprimer, après ladite étape d'insérer ou d'effacer, la limitation d'accès à l'enregistrement correspondant à la prochaine clé suivante.

7. Système selon la revendication 6 comprenant, pour insérer ledit enregistrement en cours, des moyens pour déterminer si la clé associée à l'enregistrement en cours peut être insérée dans ledit noeud de feuille et, lorsque la clé ne peut pas être insérée ainsi, des moyens pour fournir une indication que la structure arborescente d'index doit être changée, des moyens pour modifier l'arbre d'index afin d'ajouter un noeud pour recevoir ledit enregistrement en cours, et pour supprimer l'indication du changement

de la structure arborescente avant l'insertion de la clé et de l'enregistrement en cours.

8. Système selon la revendication 6 comprenant, pour effacer ledit enregistrement en cours, des moyens pour déterminer si la clé associée à l'enregistrement en cours est le seul enregistrement dans ledit noeud de feuille et, lorsqu'il en est ainsi, des moyens pour fournir une indication que la structure arborescente doit être changée, des moyens pour effacer la clé et l'enregistrement en cours, des moyens pour modifier la structure arborescente d'index de manière à supprimer le noeud, et pour supprimer l'indication du changement de la structure arborescente d'index.

DATA

EMPLOYEE NAME	EMPLOYEE NUMBER
ANDREW	1
BAKER	3
CHESTER	8
EDGAR	15
EDWARDS	24
HOWELL	7

10

FIG. 1

12

	28	30	32
14	ADDRESS	EDWARDS	24
16	ADDRESS + 2	EDGAR	15
18	ADDRESS + 4	ANDREW	1
20	ADDRESS + 6	HOWELL	7
22	ADDRESS + 8		
24	ADDRESS + 10	BAKER	3
26	ADDRESS + 12	CHESTER	8

FIG. 2

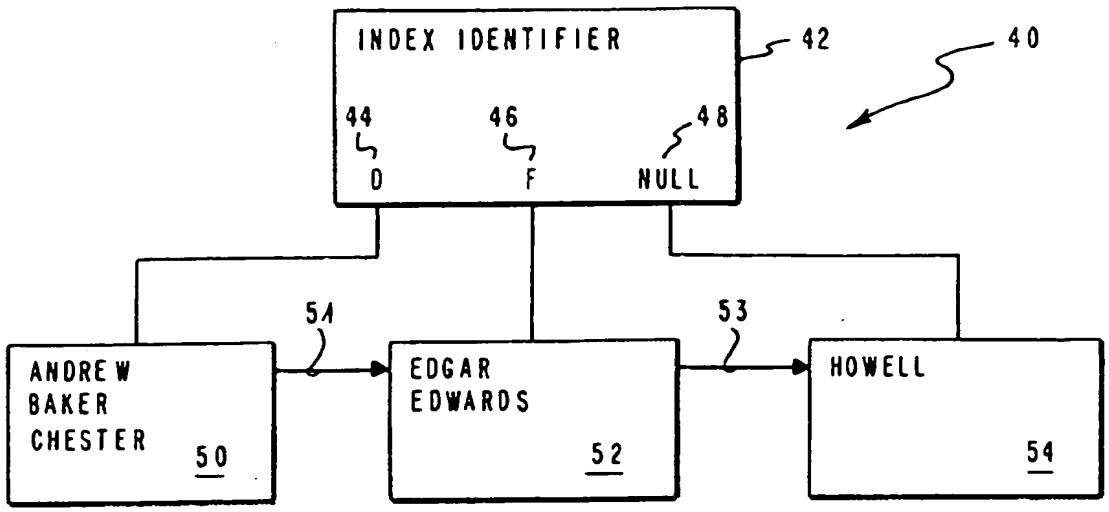
INDEX IDENTIFIER
42
40
44
D
46
F
48
NULL
51
53
ANDREW
BAKER
CHESTER
50
EDGAR
EDWARDS
52
HOWELL
54

FIG. 3

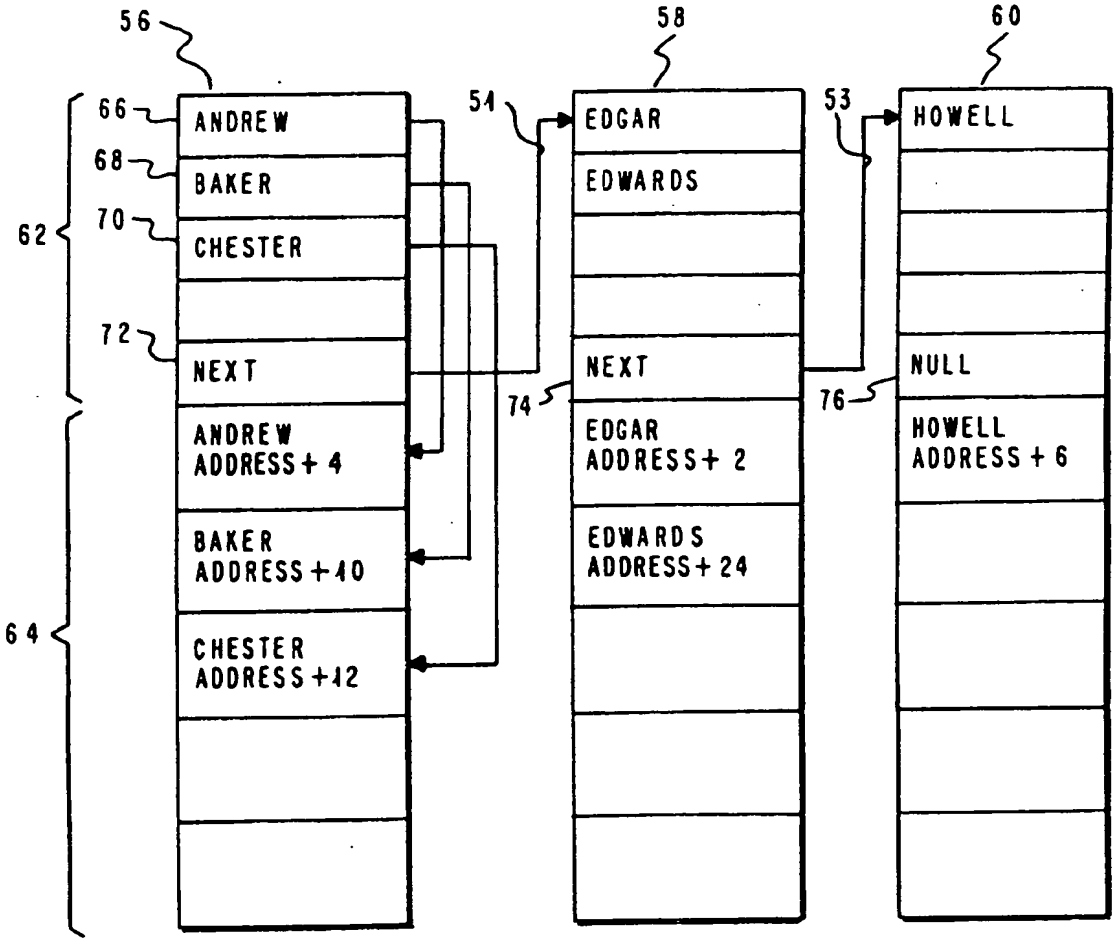
56
58
60
66
68
70
72
62
64
51
53
74
76
ANDREW
BAKER
CHESTER
NEXT
ANDREW ADDRESS + 4
BAKER ADDRESS + 40
CHESTER ADDRESS + 12
EDGAR
EDWARDS
NEXT
EDGAR ADDRESS + 2
EDWARDS ADDRESS + 24
HOWELL
NULL
HOWELL ADDRESS + 6

FIG. 4

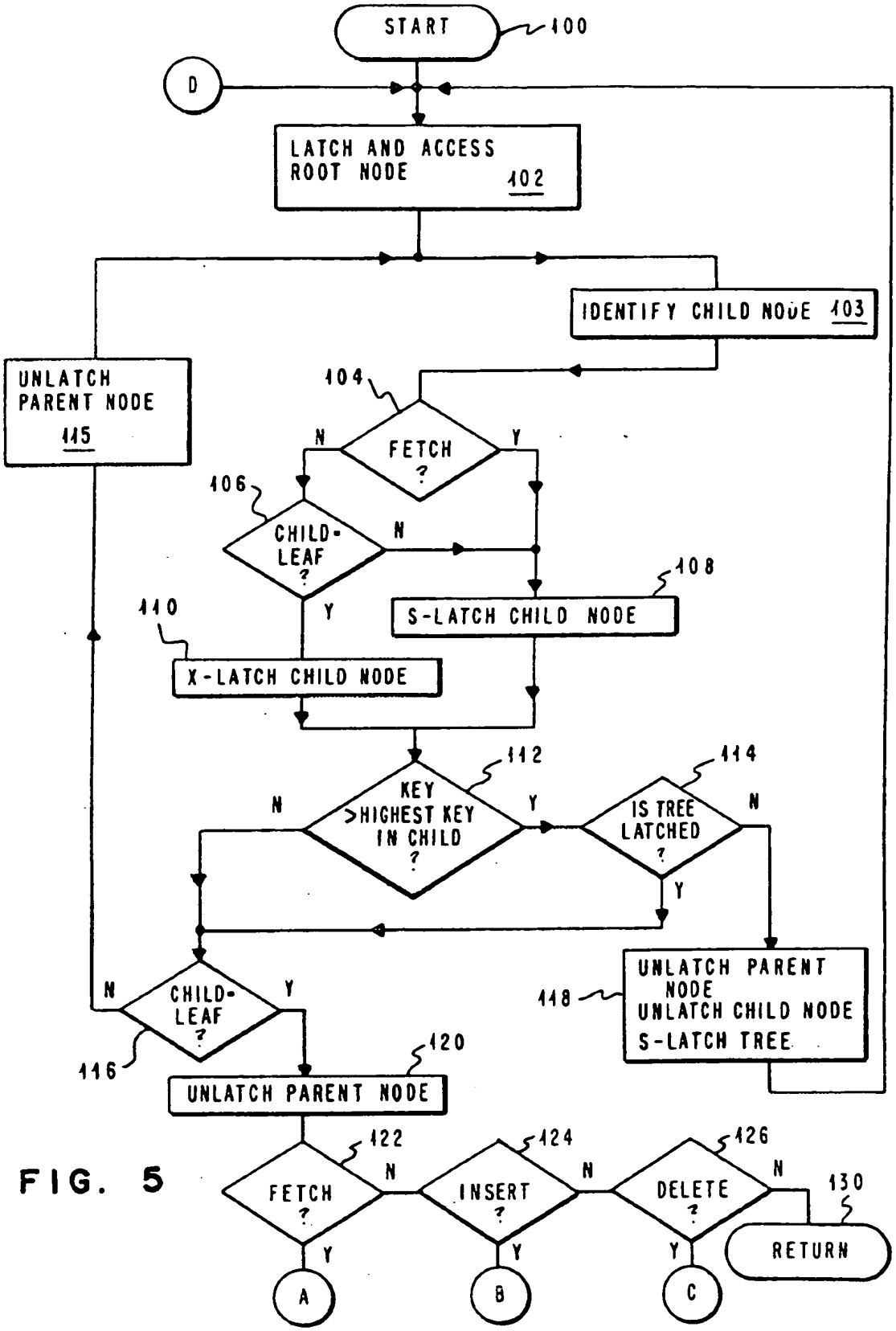
START
100
D
LATCH AND ACCESS ROOT NODE 102
IDENTIFY CHILD NODE 103
UNLATCH PARENT NODE 115
104
FETCH ?
N
Y
106
CHILD-LEAF ?
N
Y
108
S-LATCH CHILD NODE
110
X-LATCH CHILD NODE
112
KEY >HIGHEST KEY IN CHILD ?
N
Y
114
IS TREE LATCHED ?
N
Y
CHILD-LEAF ?
N
Y
116
118
UNLATCH PARENT NODE
UNLATCH CHILD NODE
S-LATCH TREE
120
UNLATCH PARENT NODE
122
FETCH ?
N
Y
A
124
INSERT ?
N
Y
B
126
DELETE ?
N
Y
C
130
RETURN

FIG. 5

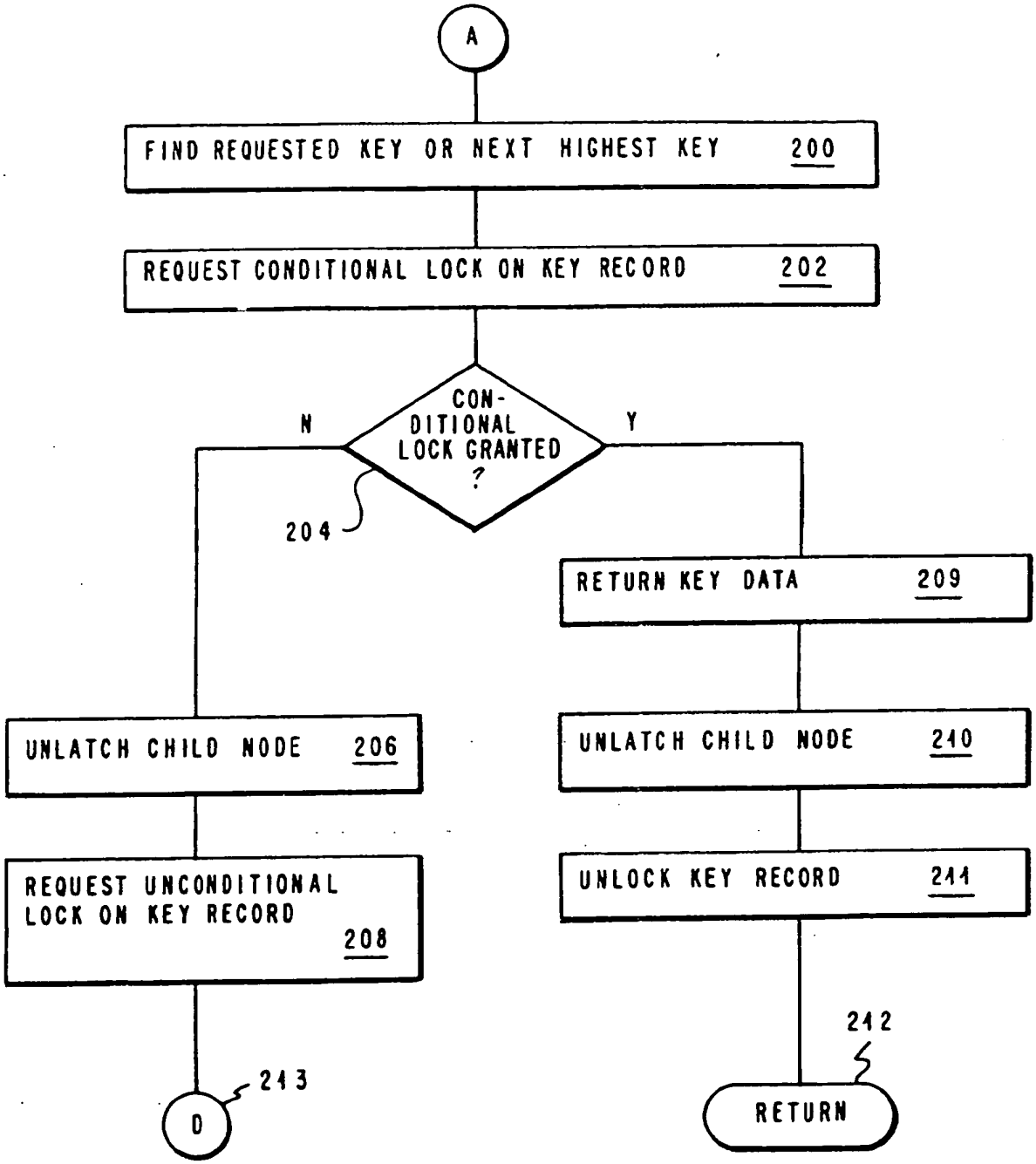
A
FIND REQUESTED KEY OR NEXT HIGHEST KEY 200
REQUEST CONDITIONAL LOCK ON KEY RECORD 202
CON-DITIONAL LOCK GRANTED ?
204
N
Y
RETURN KEY DATA 209
UNLATCH CHILD NODE 206
UNLATCH CHILD NODE 210
REQUEST UNCONDITIONAL LOCK ON KEY RECORD 208
UNLOCK KEY RECORD 211
212
213
D
RETURN

FIG. 6

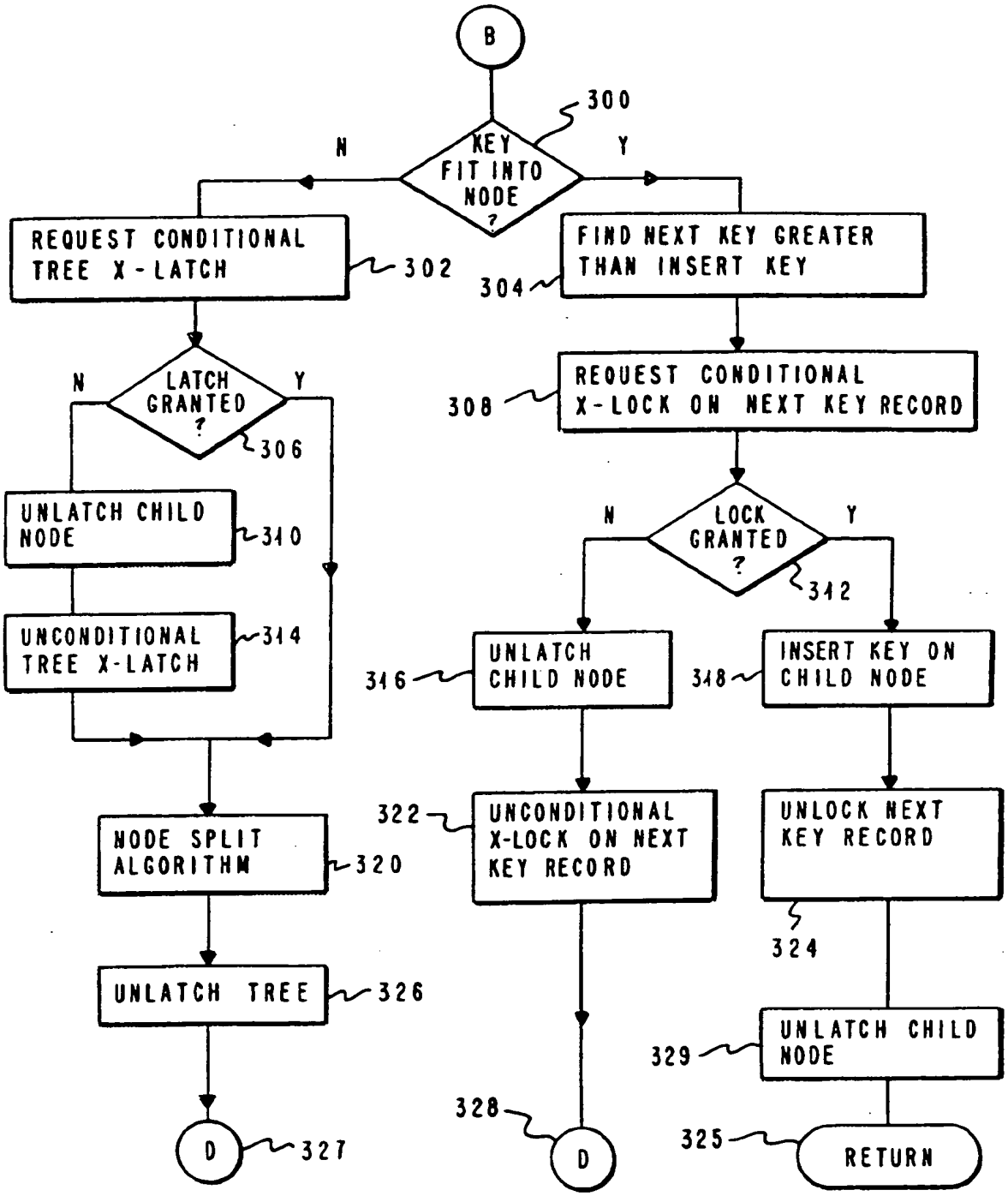
B
300
KEY FIT INTO NODE ?
N
Y
REQUEST CONDITIONAL TREE X-LATCH
302
FIND NEXT KEY GREATER THAN INSERT KEY
304
LATCH GRANTED ?
306
N
Y
UNLATCH CHILD NODE
310
UNCONDITIONAL TREE X-LATCH
314
REQUEST CONDITIONAL X-LOCK ON NEXT KEY RECORD
308
LOCK GRANTED ?
342
N
Y
UNLATCH CHILD NODE
346
INSERT KEY ON CHILD NODE
348
NODE SPLIT ALGORITHM
320
UNCONDITIONAL X-LOCK ON NEXT KEY RECORD
322
UNLOCK NEXT KEY RECORD
324
UNLATCH TREE
326
UNLATCH CHILD NODE
329
D
327
328
D
325
RETURN

FIG. 7

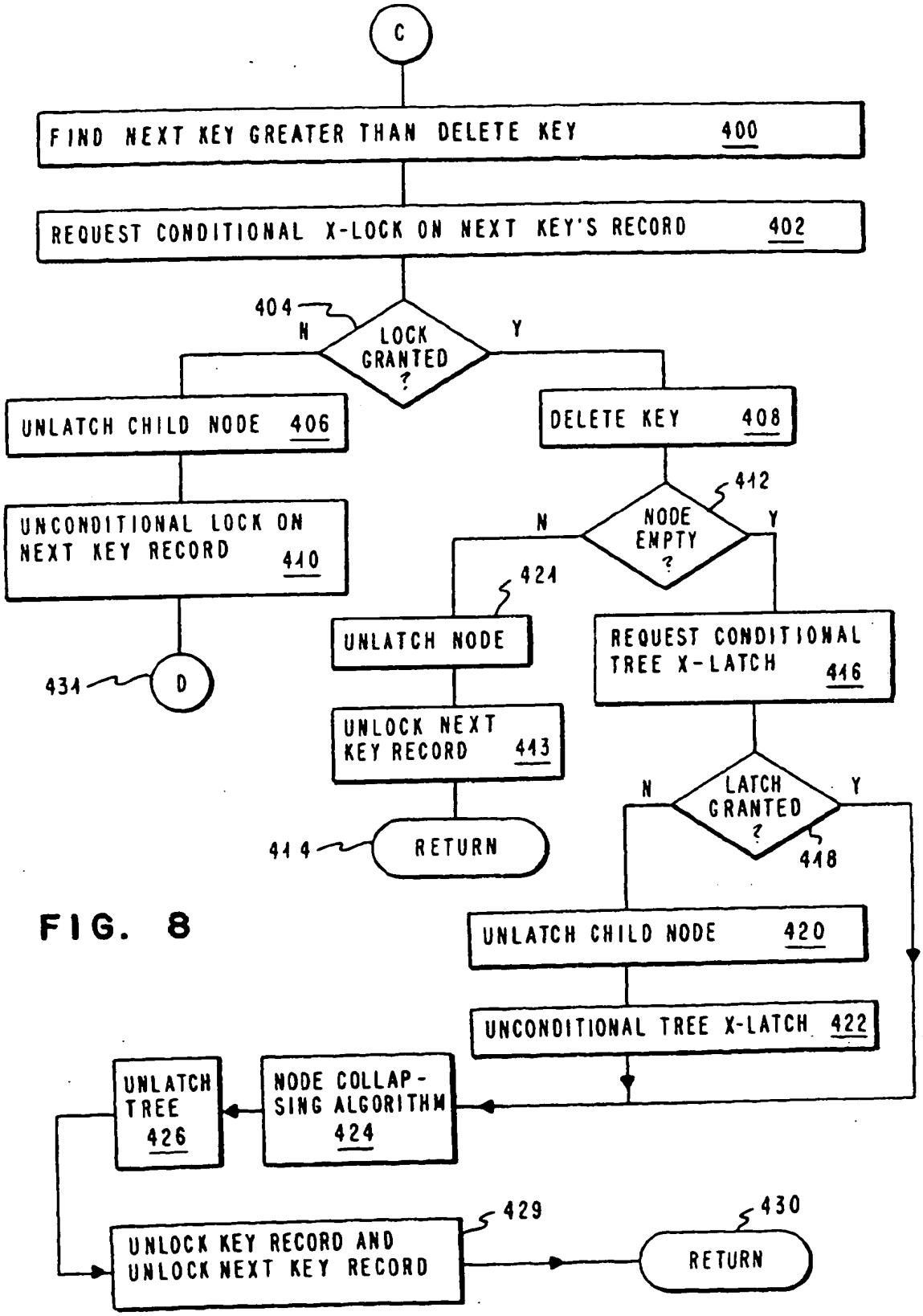
C
FIND NEXT KEY GREATER THAN DELETE KEY 400
REQUEST CONDITIONAL X-LOCK ON NEXT KEY'S RECORD 402
404
N
LOCK GRANTED ?
Y
UNLATCH CHILD NODE 406
DELETE KEY 408
UNCONDITIONAL LOCK ON NEXT KEY RECORD 410
412
N
NODE EMPTY ?
Y
421
UNLATCH NODE
REQUEST CONDITIONAL TREE X-LATCH 416
431
D
UNLOCK NEXT KEY RECORD 443
N
LATCH GRANTED ?
Y
418
414
RETURN
UNLATCH CHILD NODE 420
UNCONDITIONAL TREE X-LATCH 422
UNLATCH TREE 426
NODE COLLAPSING ALGORITHM 424
429
UNLOCK KEY RECORD AND UNLOCK NEXT KEY RECORD
430
RETURN

FIG. 8

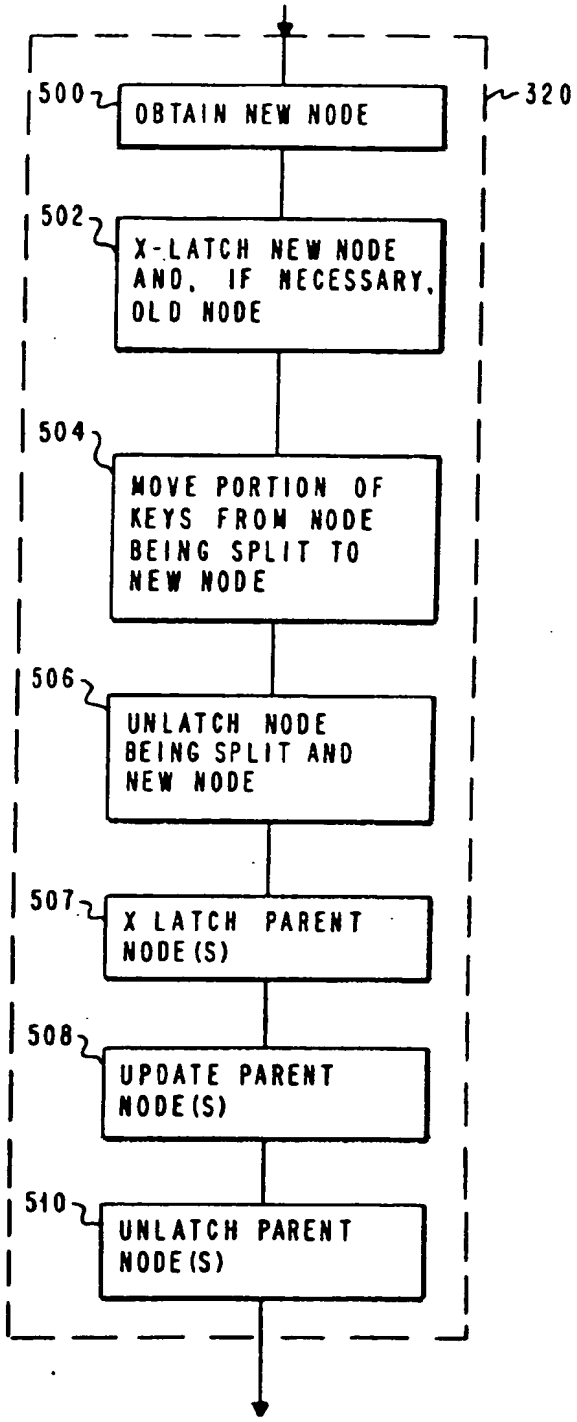
500
OBTAIN NEW NODE
320
502
X-LATCH NEW NODE AND, IF NECESSARY, OLD NODE
504
MOVE PORTION OF KEYS FROM NODE BEING SPLIT TO NEW NODE
506
UNLATCH NODE BEING SPLIT AND NEW NODE
507
X LATCH PARENT NODE(S)
508
UPDATE PARENT NODE(S)
510
UNLATCH PARENT NODE(S)

FIG. 9

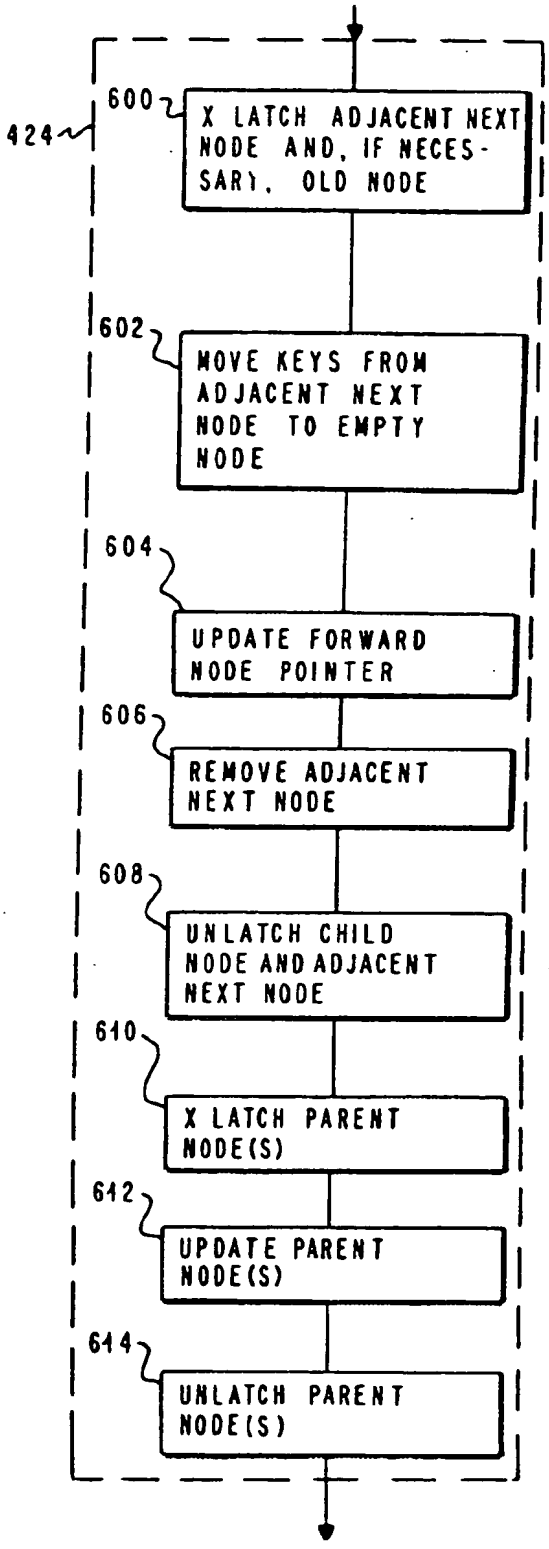
424
600
X LATCH ADJACENT NEXT NODE AND, IF NECESSARY, OLD NODE
602
MOVE KEYS FROM ADJACENT NEXT NODE TO EMPTY NODE
604
UPDATE FORWARD NODE POINTER
606
REMOVE ADJACENT NEXT NODE
608
UNLATCH CHILD NODE AND ADJACENT NEXT NODE
610
X LATCH PARENT NODE(S)
612
UPDATE PARENT NODE(S)
614
UNLATCH PARENT NODE(S)

FIG. 10